Measured Boot and Remote Attestation

Motivation

EVE system has been built with security at the core of its design. One of its SECURITY principles is that, EVE should be trustworthy, and it should provide a deterministic way to measure its software layers, right from firmware, all through bootloader, kernel and user-space applications. It should also provide a mechanism to report these measurements to a third-party for attestation. This is to provide a verifiable software environment to launch user applications, i. e. the Edge Container Objects. The concept of measured boot is not new. For example, mobile phones use measured boot and attest to a verifier, before initiating a payment transaction. Blockchain smart oracles at the cyber-physical edge, have to prove their software stack as trustworthy, before injecting events into smart contracts. This is a common requirement for distributed systems in general, but it becomes even more important for geographically remote systems like IoT Edge gateways, as there is no physical perimeter security for these Edge gateways. Following section describes the unique operational requirements of Edge gateways, and the attack possibilities associated with them:

1. In some deployments, EVE devices operate in geographically remote locations e.g. top of wind turbines or in the mid-ocean oil extraction zones, where manual access to the device is limited. In such deployments, the only way to get operational information about EVE is by EVE itself shipping the operational state to the controller. Thus providing visibility of the operational status of the device to the controller at any time is important. Even if there is an attack and the software running on EVE has been modified, it is very critical that EVE itself detects it and reports it to the controller.
2. In some deployments, Internet connectivity and power supply are not reliable. There can be intermittent loss of connectivity between EVE and Controller, and EVE is expected to continue operating with the last-known configuration till it is able to reach the Controller again. There can be power outage as well, which means, once power is back, EVE is expected to boot up and get back to its last operational state and resume its Edge Container Instances (even if the connectivity is still down)
3. Some of the EVE devices are also deployed in cases where the device is prone to security attacks, e.g. smart poles, Industrial factory floor etc. Therefore these EVE devices can be easily attacked by inserting USB drives, or the device itself can be physically hijacked to a different location for possible extraction of user data.
4. EVE software upgrades are done remotely from the Controller. Ensuring integrity of the upgrade in a zero-trust environment is a challenge (i.e. How do we know if EVE device is running the correct version indeed ?)

These challenging environmental conditions and deployment requirements bring in their own set of security attack possibilities:

1. There could be physical attacks, e.g. booting from a USB key with modified software, installing firmware rootkits, or by altering hardware configuration by adding unauthorized PCI peripheral or removing existing PCI device
2. There could be attacks over the network, e.g. modifying OS partitions to boot a different (potentially modified version of EVE) to run malicious programs
3. There could be attempt to "steal" the device, and operate it in a different location, for off-line hacking, in an attempt to decrypt the encrypted volumes on the disk, by booting a different software
4. There could be a malicious software version booting up and pretending to be the legal version expected by the Controller, and thus extracting all the latest configuration from the Controller, which might contain secretive information like cloud storage credentials, sensitive user-data such as cloud config for a Edge Container Instance etc.

That is the problem to solve - EVE should maintain operational availability in these diverse conditions, and at the same, EVE should also support a security framework to detect and mitigate these security challenges. i.e.

1. Measure the boot chain of EVE
2. Detect any discrepancies in the boot chain and disallow access to sensitive resources in EVC(like the credentials)
3. Allow EVE to access encrypted volumes of disk, as long as the measurements don't change
4. Have self-locking mechanism to restrict access to encrypted volumes in case of a change in the boot chain, even during offline operation.

That is what this proposal is about: Measuring the boot chain of EVE, and allowing access to select resources in the Controller only on the basis of attestation of these measurements. Since any software can be potentially modified, such measurement architectures typically use a hardware based root of trust (HRoT) or a Trusted Execution Environment(TEE). Here we present a solution based on Trusted Platform Module as the Hardware Root of Trust.

At a high level, the solution is to

1. Implement recommendations of TCG Remote Attestation Protocol TAP - with EVE as the attester and EVC as the verifier. i.e. use TPM to measure the booting sequence using Platform Configuration Registers (PCR)
2. On top of the TCG solution, propose a mechanism for self-locking: Seal the decryption key for the encrypted volumes using PCRs (for self-locking during offline/tampered conditions). This is done to address unique operational requirements of EVE at the Edge.

3. Save the encrypted volume key with the Controller during the upgrade.

Following sections describe the details, starting with a review of existing solutions in this space.

Measured Boot vs Secure Boot

Secure Boot is a security standard where each software layer in the booting sequence measures the next layer, and starts the next layer only if the signature of the software is verified by a certificate rooted to one of the known certificates embedded in the firmware by OEM. Software updates are done through uploading signatures and certificates of the new binary. Since each layer starts the next software layer only if the signature verification is successful, boot process is stopped when one of the stages fail the signature verification. For secure boot to work, all the software layers should be capable of measuring and verifying the next software layer.

Measured Boot, is another security standard, where the measurements are recorded into TPM PCRs but the boot process can be allowed to complete. The measurements can be later read back from TPM and the Audit log of all the measurements can also be fetched. Thus measured boot does not make any judgement about the integrity of the boot stages, but gives opportunity for an external verifier to inspect the TPM Audit Log and PCR values in a detailed manner to see which components were measured, what their measurements are etc., and make a decision.

Please note: There are some measured boot solutions which behave more like secure boot, there are some secure boot solutions which behave more like measured boot, but the reader is advised to stick to the above definitions, for the purpose of discussions in this document.

Secure Boot	Measured Boot
Core Root of Trust is certificate embedded in Firmware by OEM	Core Root of Trust is Trusted Platform Module (TPM)
Does not require connectivity to any remote verifier to establish chain of trust.	verifier needs to inspect TPM PCR values before booting process can be trusted
Any new software needs to be signed by a certificate rooted to OEM certificates	An independent trusted third-party (an verifier), can match the PCR measurements against its expected values
Chain of trust is established by each software layer verifying the next software layer before handing the control over (in the boot chain)	Chain of trust is established after going through TPM Event Log and PCR values
Device will become inaccessible if one of the software layers fail validation since booting process will halt. Good for devices with human presence like smartphones, laptops etc. But poses difficulty for remotely managed systems like IoT Edge gateways, where manual access to the device is limited. Since IoT gateways are managed through remotely located controllers, losing connectivity will severely impact visibility into the system.	Works well for remotely managed systems. Device will be accessible, and controller can still reach the system, and get visibility into the system.

As you can see, both the standards have their own merits and demerits when considering security requirements for a remotely managed EVE system. i.e.

1. EVE needs the ability of Secure Boot to establish chain-of-trust without controller (so that EVE can continue to function even during intermittent Internet blackouts)
2. At the same time, EVE needs the ability of Measured Boot, where the boot process is completed, and not halted in between (To provide visibility to the EVC)

Therefore, one needs to look at how we can meet these two seemingly conflicting goals. In the following sections, we present a solution mostly based on the principles of measured boot, but also considering the offline operational requirement of EVE. In the end, we also present a hybrid model, where we can add some features from Secure Boot, to provide more security against firmware Rootkits.

In the following sections, we first present the key capabilities of TPM, and then describe how these capabilities can be used in EVE system to arrive at the solution.

Trusted Platform Module as the Silicon RoT

Trusted Platform Module(TPM) supports many crypto functions. Notably the “PCR Extend” and “Seal” operations are used in popular measured boot architectures. Let's take a quick look at these commands.

PCR Extend

TPM can be asked to perform a **“PCR Extend”** command, where a particular hash value would be added to the existing hash value in a Platform Configuration Register(PCR), and the resultant hash value can be stored back in the same PCR. i.e.

$$\mathbf{PCR\_Content_{New} = Extend\ (PCR\_Content_{Current,}\ New\_Measurement)}$$

One can only extend the current value in the PCR with a new hash value, and the existing contents can not be overwritten. This provides us these key capabilities:

1. The order of measurements - Final value will be the same if and only if the measurements are done in the same order
2. Final PCR value captures the whole history of measurements - useful in quick validation of final states against expected state
3. Deterministic - If one repeats the same history of measurements, he will end up in the same final PCR value - Useful in validation of a change in one of the input sequences

Seal

TPM can seal a given secret information against the current PCR values, through a TPM command called **"Seal".** Once sealed, the information can be read back only through an unseal command, which will succeed only if those PCRs hold the same set of values as they were during the sealing operation. In other words, if those PCR values aren't the same, the secret can not be recovered.

Using these two TPM capabilities, one can build powerful solutions to measure and validate the software state of a given system. The measuring process is called Measured Boot, and the method of getting the measurements verified and attested through a third-party is called Remote Attestation

Measured Boot is a method where each of the software layers in the booting sequence of the the device , measures the layer above that, and extends the value in a designated PCR. e.g. BIOS measures various components of Bootloader and stores these values in PCRs 0-7. Likewise, bootloaders can measure the Linux kernel and store the measurements in PCRs 8-15. The Linux kernel has a feature called Integrity Measurement Architecture (IMA), where various kernel executables/drivers can be measured and stored in PCR 10.

During these extend operations, the extend operations are recorded by BIOS and Bootloader, in a special firmware table, called the TPM Eventlog table, and this table is handed over to the operating system during OS takeover. By playing the same sequence of extend operations recorded in a given TPM Event Log, one can check if the final PCR values match, and if so, then the Event Log(and hence the software layers) can be trusted.

Measuring using PCRs, Verifying using TPM Event Log

Based on the above constructs, we present a solution to measure and attest software integrity of the device . Just for recap, EVE is the open-source software from LF-Edge for Edge Virtualization, running on IoT Edge gateways. EVC is the controller for managing these EVE instances. Adam under LF-Edge is an open source implementation of one such EVC. The APIs between EVE and EVC are specified in EVE API specification. In the context of remote attestation, the EVC is the attesting authority and EVE reports its measurements for attestation.

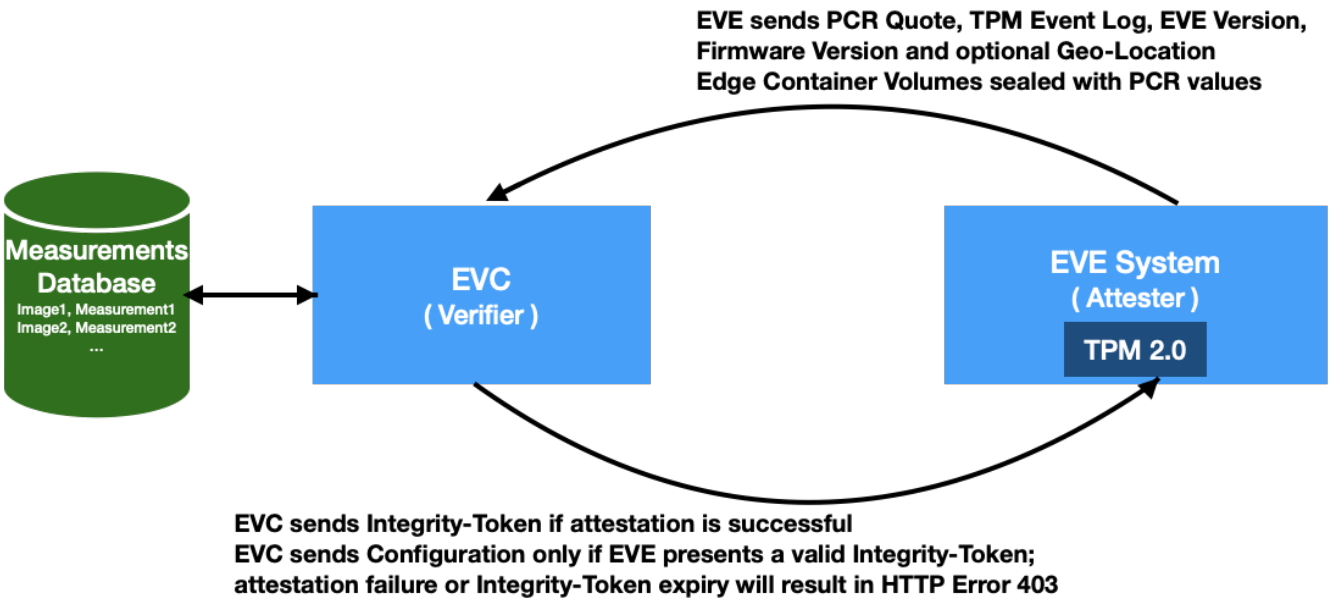

Fig 1. Role of EVC as the verifier, and EVE as the attester

Introduction to Foundational Concepts

1. A new device state, "**Unknown Update Detected"** (Abbreviated as **UUD**) will be introduced in EVC, to notify that measurements of software running on a given device didn't match expected measurements.
2. EVC maintains a central database of all the supported EVE software images, and their hash values, indexed with the EVE image version tag
3. EVC also maintains a central database of mostly used BIOS firmware images, their signatures, and the certificate provided by the BIOS vendor for validating the signatures, indexed by a combined tag of BIOS version string + Manufacturer
4. On receiving attestation request EVE, attestation service module checks PCR Quote against the baseline value. If there is a change, the device is marked as "**UUD".** When there is no software change, the PCR quote is expected to be the same across reboots. However, after an EVE software upgrade, it is expected that the baseline value will change for the PCR values. However, after comparing the reported values with the expected values(since EVC knows about the new image version and its hash values), EVC makes a decision: If the reported values match, the baseline for the device is updated. If they don't, the device is marked as UUD.

5. An optional feature, **"Location-Lock"** may be introduced in EVC, to additionally check the Geo-Location reported by device, and flag if the location has changed since its last-seen location. This feature may be critical in some deployments, where a given device may be mounted permanently, and any change in its physical location should be flagged, and optionally considered along with software measurements to conclude if the device is trusted. In this regard, another new device flag, **“Unknown Movement Detected”** (Abbreviated as **UMD**) will be introduced in EVC, to indicate that a change in the location of the device has been detected.
6. If the device is not in “UUD" or "UMD" state, EVC includes all the latest configuration in reply to config request from EVE.
7. If edge-node is in “UUD” or "UMD" state, any config request from EVE will be responded with response code 403 - Forbidden. This is to protect any new sensitive images/credentials from getting exposed to the compromised device. The response from EVC in such cases will carry an error code to indicate that there was an attestation failure, and hence partial configuration is being sent. EVE, upon receiving such error codes MUST schedule re-attestation immediately.
8. If EVE reboots with a different software image from the configured version, EVC should be able to detect as quickly as possible and force attestation. To this effect a new token is introduced, called **"Integrity Token".** This token is a random nonce that will be stored in EVE in process memory, which is valid only for the current boot session. Thus the key property of this Integrity Token is that it clears up automatically after every reboot. Therefore, if EVE reboots, it will automatically cause re-attestation, since there will not be any Integrity-Token to present to the Controller. This token is sent during attestation requests, and if the attestation is successful, the provided token value is sealed against PCR values. This is usually implemented by storing the token inside an encrypted folder, with the Volume Storage Key protected by TPM with Seal operation.
9. Every configuration request from EVE will include this Integrity Token. If there is a token mismatch, HTTP code 403 - Forbidden will be sent to the device as the response, indicating that device should do re-attestation.
10. Frequency of attestation: the device will be required to periodically attest itself via attestation requests. It is up to the EVC to schedule the frequency of attestation. Whenever EVC replies with Error 403, EVE MUST re-attest. EVC can trigger this either periodically (say every few hours) or when it sees any discrepancy in the Integrity-Token.

Protecting ECO Instances by Sealing ECO Volumes

All the ECO Volumes would be inside the encrypted folder (/persist/vault), with the Volume Storage Key sealed inside TPM, against a set of PCR values. Without access to this key for the encrypted volume, the encrypted folders can not be accessed. Essentially this means that the contents of the user-provisioned volumes are sealed against PCR values.

If EVE or BIOS image was changed during boot, then unsealing of this storage will fail, since the PCRs have changed. Since ECO Volumes are inside the sealed storage, ECO volumes are locked till re-attestation happens with the EVC.

To implement this, ECO volumes would be inside the encrypted folder (/persist/vault), with the Volume Storage Key sealed inside TPM.

To provide some flexibility to the device administrator, when deploying ECOs on the EVE platform, there will be 3 security modes for app deployment:

1. **No-Lock Mode** - ECO Volumes will not be in encrypted folder (not recommended, unless there is a compelling need to run any ECO unconditionally e.g. service apps like SD-WAN, DHCP Server)
2. **Software Lock Mode** - ECO Volumes will be in encrypted folder - with key from TPM, sealed with PCRs (**the default option**)
3. **Software Lock + Location Lock Mode** - ECO Volumes will be encrypted folder - with key from TPM, sealed with PCRS + a key provided after Geo-Location check from the EVC. This will work only when device is connected to EVC

Offline vs. Online checks

With the approach to seal the key(s) needed for fscrypt under the TPM, this means that if the PCR values have not changed (e.g., a normal reboot, or a reboot due to a power outage) the device will be able to unseal those keys and start the ECOs without any controller connectivity.

After a BIOS and/or EVE update, the PCR values will presumably change so the unsealing will fail. In that case the device needs to perform remote attestation with the controller (with automatic and/or user acceptance of the new measurements) before it can retrieve the key from the controller. Note that as part of EVE update we already require this connectivity to verify that the update is fully working. Once retrieved the device will again seal that key under the current PCR values.

This means that the controller will need to store the fscrypt key(s) so it can hand it back to the device . But we should be able to encrypt them under a separate private key from TPM so that even if the controller is compromised the fscrypt keys can not be extracted from it.

The offline approach means that if there is something else wrong, such as IP geolocation doesn’t match, the applications are likely to start before the controller can raise a flag and potentially disable the applications.

And since this is measured boot it means that if some firmware or EVE component has been compromised, EVE will still boot and attempt remote attestation. Such a compromised device might be running some root-kit which can access the adapters on the device , but applications will not start. Thus this implicitly assumes that the security of connected systems is based on credentials stored in the application instances and not merely by having the physical connectivity to some serial port or Ethernet network.

Updated EVE Startup State Transition

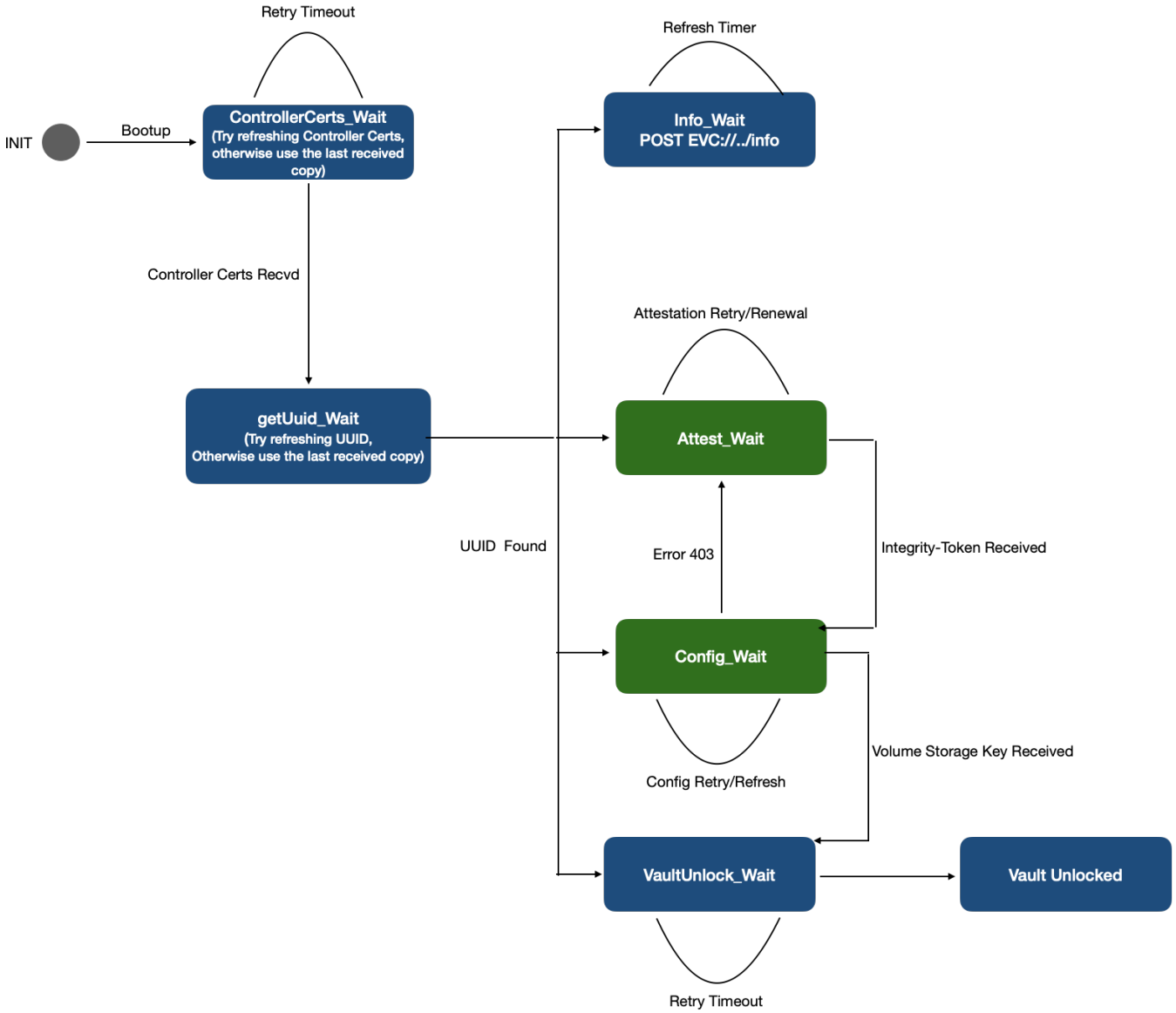

Fig 2. State Diagram for EVE Startup, with Attestation enabled

During Startup, EVE performs the following logical steps:

1. Fetches certificates used by EVC. e.g. X.509 certificate chain used in signing AuthContainers. If EVC is not reachable, goes with the cached copy of the certificates.
2. Tries to retrieve UUID provisioned in the EVC for this EVE instance. If EVC is not reachable, goes with the cached copy of the UUID last received from EVC.
3. Attempts to unseal the key in TPM. If unsuccessful, waits for encrypted Volume Storage Key from EVC, published post-attestation by EVC through configuration response.
4. Whenever HTTP Error 403 is received from EVC, starts a new attestation cycle with EVC:
    a. First EVE requests EVC for a nonce to include in the PCR quote, as a freshness proof.
    b. Interacts with TPM Mgr to prepare PCR quote with this nonce value as user-data.
    c. Then, it generates a random value to propose as the Integrity-Token.
    d. Finally sends attestation request with the following fields:
        i. The new Integrity-Token value
        ii. PCR Quote
        iii. TPM Event Log
        iv. Image version (EVE and Firmware)
5. Whenever attestation response is successful, populates the Integrity-Token (included in the EVC's reply to attestation request), in a memory mapped file (not on the disk). Since Integrity-Token is volatile, when the device reboots, this automatically triggers re-attestation.
6. If attestation fails, attestation is re-attempted periodically.
7. Configuration is requested in periodic intervals, with the Integrity-Token(or without that, which will trigger error 403).
8. If config response contains attestation error (i.e. HTTP error 403), triggers attestation request immediately.
9. Even if attestation is successful, EVC can choose to invalidate Integrity-Token periodically (say every few hours), to re-trigger attestation. Since Integrity-Token is invalidated, next config request from EVE will receive the same HTTP 403 error code in this case, which will re-trigger attestation.

The next section presents the same steps, but at more functional level, with interaction among various EVE microservices.

Module Level Interaction - Startup Sequence (Reboot with no change) - No EVC connectivity required

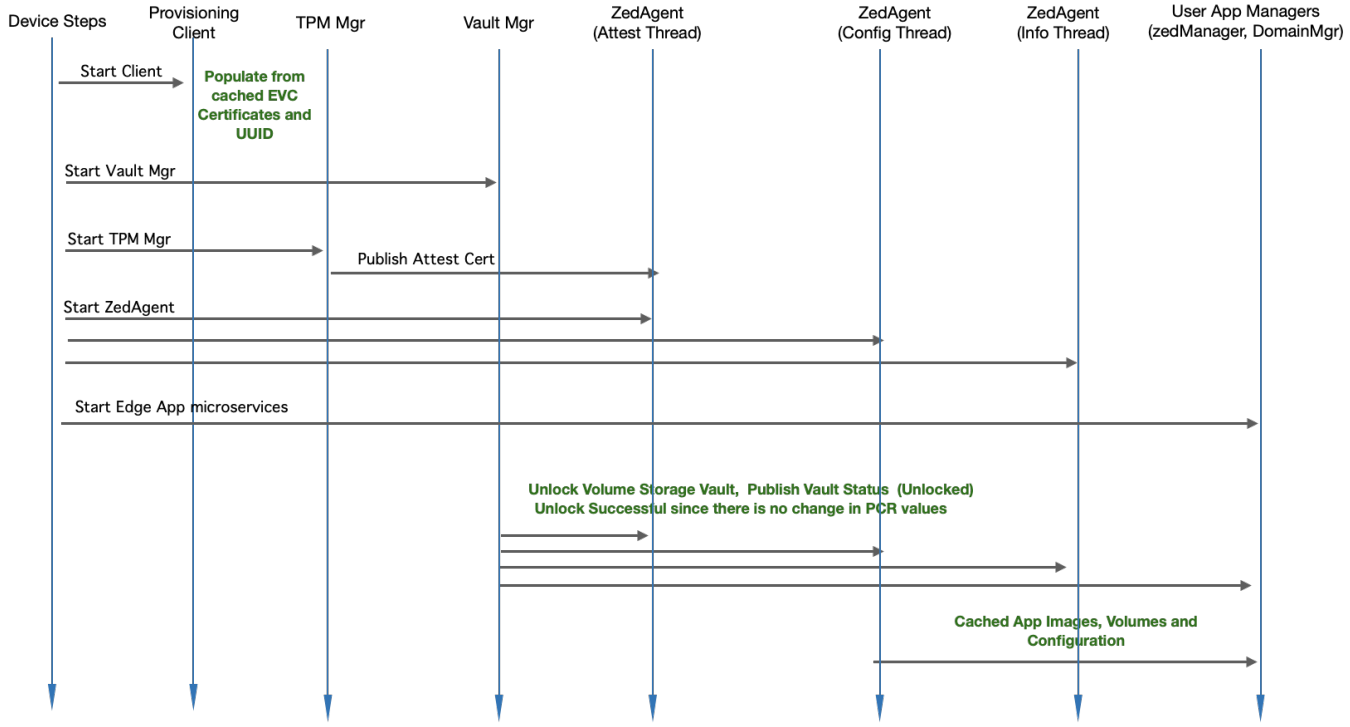

Fig 3. Module Level Interaction in EVE - Startup Sequence (Reboot with no change; works without EVC connectivity)

1. Device-steps starts client.go (the provisioning client) which will check and do the following:
    a. If EVC is not reachable, uses certificates from the last download
    b. If EVC is not reachable, uses UUID already present in /config/uuid
2. Device-steps starts Vault Mgr
    a. Vault Mgr retrieves the master decryption key from TPM with Unseal operation. Unseal is successful since the PCR values are the same since last reboot.
    b. Unlocks /persist/vault with the Volume Storage Key.
    c. Publishes vault status (unlocked)
3. Device-steps starts TPM mgr
    a. TPM manger retrieves the attestation certificate and publishes to Zedagent
    b. Waits for Quote requests on IPC channel from Zedagent
4. Device-steps starts Zed Manager, Domain Mgr microservices - these services are responsible for launching the ECOs
5. Device-steps starts Zedagent (and Zedagent starts 3 concurrent tasks: attest, info and config)
    a. Configuration task, since EVC is not reachable, uses the configuration from the last download, and publishes to Zed Mgr and Domain Mgr
    b. Since the Vault is unlocked, Domain Mgr can access Edge App Images, and hence launches the ECOs.

Module Level Interaction - EVE Startup Sequence (Reboot with a change)

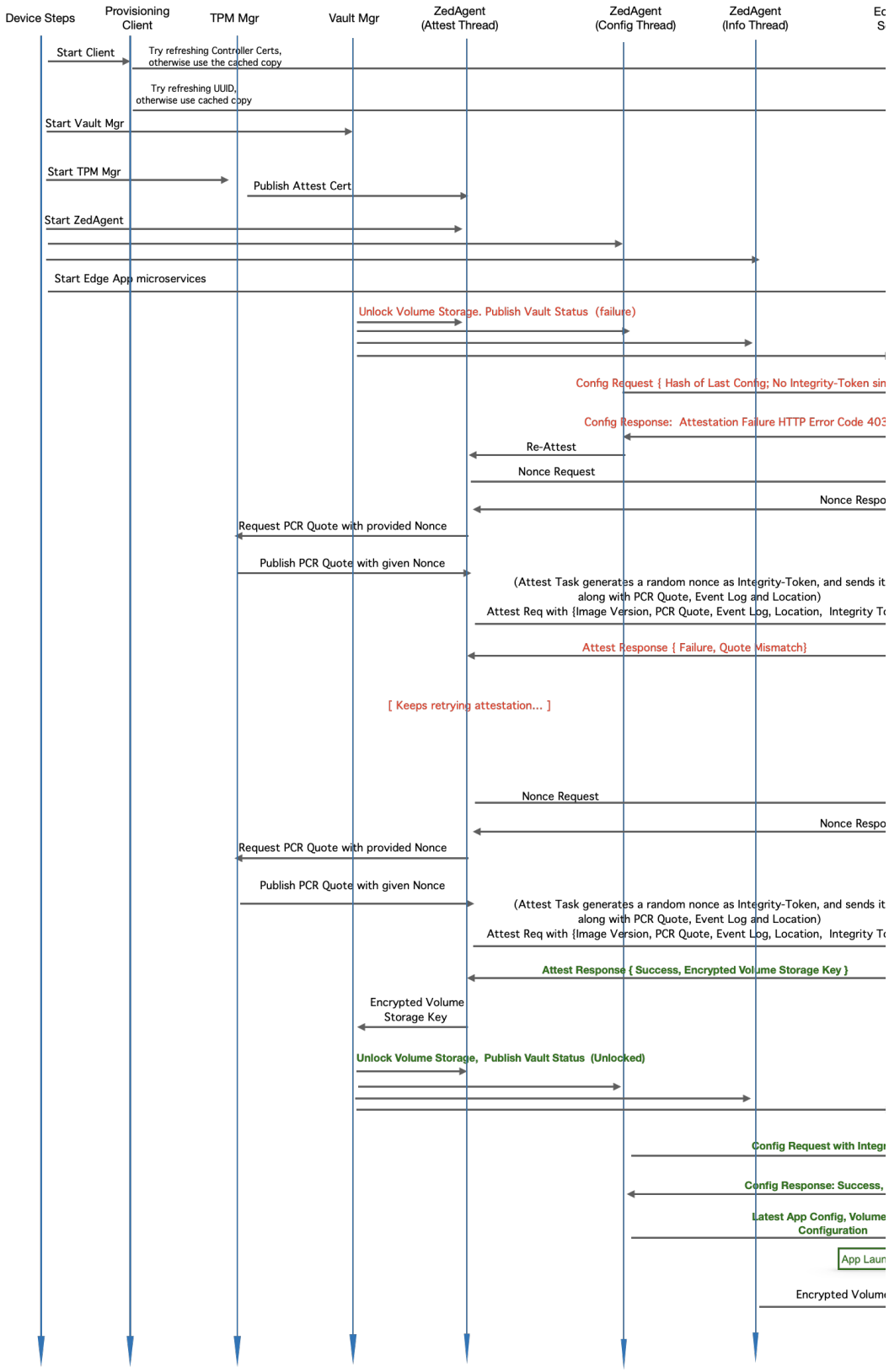

Fig 4. Module Level Interaction in EVE - Startup Sequence (Reboot with differing PCR values)

1. Device-steps starts client.go (the provisioning client) which will check and do the following:
    a. If certificates from EVC are not yet fetched, fetches them
    b. Retrieves UUID from EVC
2. Device-steps starts Vault Mgr
    a. Vault Mgr tries to retrieve the master decryption key from TPM with Unseal operation, and Unseal operation fails since the PCR values have changed

    b. Publishes vault status (as "locked")
    c. Waits for Volume Storage Key from EVC - it will block here forever
3. Device-steps starts TPM mgr
    a. TPM manger retrieves the attestation certificate and publishes to Zedagent
    b. Waits for Quote requests on IPC channel from Zedagent
4. Device-steps starts Zedagent (and Zedagent starts 3 concurrent tasks: attest, info and config)
5. Device-steps starts Edge App microservices - e.g. Domain Mgr
    a. Attest task requests for a nonce from EVC (to prepare PCR quote)
    b. Attest task sends the nonce back to TPM Mgr and waits for PCR quote
    c. Attest task, once notified about the quote readiness, creates a random nonce value for Integrity-Token
    d. Attest task sends { Quote, Location, Event Log, Integrity-Token, Image Version } to EVC
    e. EVC, since the quote is different, tries to compare EventLog entries with its known hashes against the version reported. Since the PCR values are different, there will not be match, unless Admin has uploaded measurements for this new version. Assuming that Admin is yet to upload, attestation request will fail here.
6. EVC sends an error back to EVE, to retry attestation.
7. In the mean time, configuration task keeps requesting config from EVC (expected to fail till attestation goes through).
    a. EVC replies to configuration request with HTTP Error Code 403 - Forbidden. Indicates attestation failure (due to no or invalid Integrity Token)
    b. Config task communicates to Attest Task to re-trigger attestation
8. Admin uploads measurements for the new image version
9. The next attestation request is matched successfully against the uploaded measurements
10. EVC approves the attestation request, and responds back with the Volume Storage Key (encrypted with TPM key) for the device , and also caches Integrity-Token supplied in the attestation request in a memory mapped file
11. Vault Mgr
    a. picks up the Volume Storage Key
    b. unlocks vault
    c. populates Integrity-Token approved by EVC inside a memory mapped file
    d. Publishes new vault status (unlocked) to all the microservices
12. The next configuration request picks up the new Integrity Token, and sends configuration request along with this Integrity-Token
13. EVC verifies that Integrity-Token matches with its copy, and approves configuration request and responds with the latest configuration.
14. Domain Mgr notices the new vault status (unlocked) from Vault Mgr , and latest configuration from ZedAgent, and starts the Edge Apps
15. Info task reports the encrypted key for the encrypted volume back to EVC, for backup purposes

Module Level Interaction - EVE Startup Sequence (Initial Onboarding)

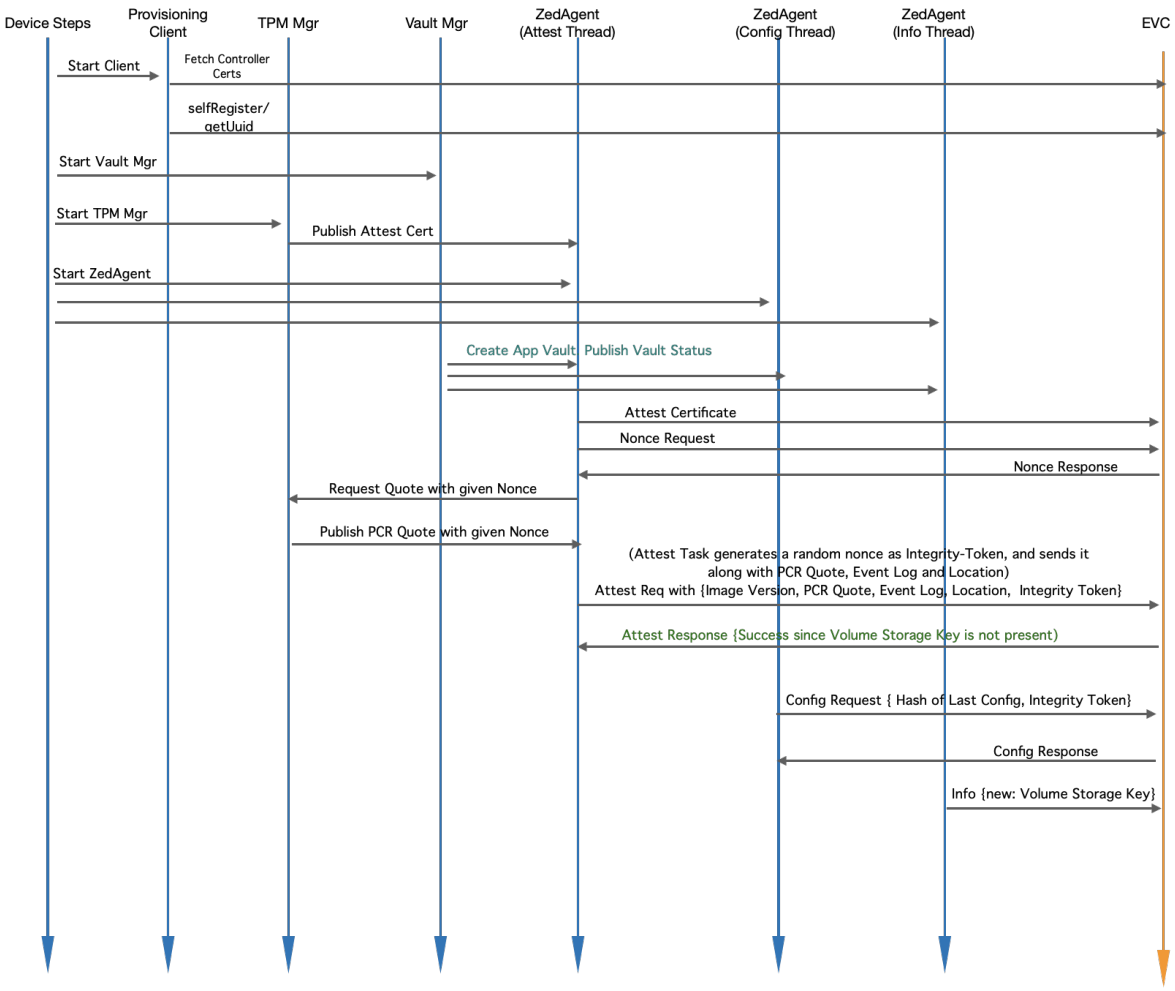

Fig 5. Module Level Interaction in EVE - Startup Sequence (Onboarding)

1. Device-steps starts client.go (the provisioning client) which will check and do the following:
    a. First fetches EVC Certificates
    b. Registers the device certificate
    c. Retrieves UUID from EVC
2. Device-steps starts Vault Mgr
    a. Vault Mgr creates new vault: /persist/vault
    b. Seals the Volume Storage Key into TPM with PCR values
    c. Publishes vault status (unlocked)
3. Device-steps starts TPM mgr
    a. TPM manger retrieves the attestation certificate and publishes to Zedagent
    b. Waits for Quote requests on IPC channel from Zedagent
4. Device-steps starts Zedagent (and Zedagent starts 3 concurrent tasks: attest, info and config)
5. Attest task,
    a. picks up the attestation certificate and publishes to EVC
    b. requests for a nonce from EVC (to prepare PCR quote)
    c. sends the nonce back to TPM Mgr and waits for PCR quote
    d. Once notified about the quote readiness, creates a random nonce value for Integrity-Token
    e. Sends { Quote, Location, Event Log, Integrity-Token, Image Version } to EVC
6. EVC, if key for the encrypted volume is not found for the EVE, approves the PCR quote as the baseline, provided the hash values match the EVE version reported.
    a. If Location-Lock is enabled, location is also approved as baseline (provided the EVE device is not in manufacturing account)
7. In the mean time, configuration task keeps requesting config from EVC (expected to fail till attestation goes through)
8. Attest task, once attestation request is approved, gets the Volume Storage Key it to Vault Mgr
9. Attest task, also caches the Integrity-Token acknowledged by EVC in a memory mapped file
10. Config task now picks up the right Integrity-Token and Config request is now accepted by EVC, and full configuration is sent back
11. Info task reports the encrypted key for the encrypted volume back to EVC, for backup purposes

Securely Upgrading EVE Software

During regular reboot cycles of EVE (without any software change), the PCR values are not expected to change, and hence with a simple check of PCR quote (with a nonce to prove freshness), EVC can make a quick check and approval on the state of software post-reboot. However, it is during software change that there are additional steps involved. To understand it better, consider following two cases:

A Legal Upgrade scenario: Due to a software upgrade triggered, EVE reboots with new software, and presents a PCR quote different from its last-reported value

An Attack scenario: A rogue software is installed, and EVE reboots with the rogue software, and pretends to be one of the supported versions of EVE, and presents a PCR quote different from its last-reported value

As you can see, in both the cases EVE presents a different PCR quote than the last-reported value. One case is genuine and should be allowed but in the other case, it is a rogue software and should be detected and reported!

Since EVE must present PCR quotes in both the cases (otherwise config request will fail with Error 403), EVC can examine PCR quote to see if it matches with PCR quote of the EVE version it thinks EVE should be running presently. In other words, for EVC to differentiate between approved vs unapproved image version, one must mark the supported EVE image versions as "approved" by entering their corresponding measurements values expected in the TCG measurement sequence (since multiple values can be extended to a single PCR). With the help of this "list of approved measurements" , EVC can co-relate the entries in TPM EventLog to confirm if the boot chain with the modified software image is trusted and expected. It is possible that a rogue software might, after reboot, not send any config request at all, and still trying to access the contents on the disk, like credentials, sensor feed data, etc from the Edge Container Volumes. But the decryption key for accessing these encrypted volumes (the Volume Storage Key), is not stored anywhere in memory or on disk, but rather it is sealed with the PCR values of the previous software version. The only other way to get access to this key is from EVC, but EVC will share this key only on successful attestation. Through such a backup of encryped material, the EVC ensures that only approved EVE images can access the protected resources like Edge Container Volumes. Since PCR quote is always generated with a nonce given by EVC from time to time, this provides freshness proof for the quote, and a rogue software can not copy PCR quote from a supported EVE version and pretend that it is the latest quote.

Putting it all together, integrity of the upgrade is established through the following steps (Fig 6):

1. While still on the old image version, EVE encrypts the Volume Storage Key using a TPM-based key, and sends it to EVC. EVE can send this information when the Volume Storage Key is created for the first time, and also can include this in the periodic info message. Sending this in periodic info message also provides additional facility of rotating the key if required.
2. After the upgrade, the EVE software presents the new Event Log, along with the PCR Quote. As mentioned earlier, the PCR Extend operations are recorded in a table called TPM Event Log Table. This is sometimes also called the Boot Log.
3. First, EVC repeats the transactions mentioned in the Event log, and the final PCR as computed from the Event Log should match the PCR Quote. If they don't match, then Event Log can't be trusted(say it was manipulated), and the device is marked as UUD. Please note: Event Log is stored in system memory and PCR Quote is generated by TPM. So PCR quote is the source of truth, pinned to HRoT.
4. Secondly, if PCR Quote matches the Event Log, then Event Log entries are compared between old Event Log and the new Event Log, and the differing entries are extracted
5. EVC maintains a central database of all the supported EVE software images, and their hash values, indexed with the EVE image version tag
6. EVC also maintains a central database of all the BIOS firmware images, their signatures, and the certificate provided by the BIOS vendor for validating the signatures, indexed by a combined tag of BIOS version string + Manufacturer
7. The differing values are compared against acceptable values stored against the given image version. Additionally, If admin has enabled "Location-Lock" feature, additionally the geo-location reported by the device is checked as well. The Geo-location reported by the device can be trusted if the software state (established through PCR quote) can be trusted.
8. If the PCR quote and the optional Geo-Location check pass, the encrypted key for the encrypted volume is given back to the device . EVE decrypts the Volume Storage Key, and unlocks the vault with the Volume Storage Key. the Volume Storage Key is now sealed into the TPM, against the current PCR values.
9. User will have an option to override the attestation decision taken by EVC, through a configuration in EVC portal, against the device

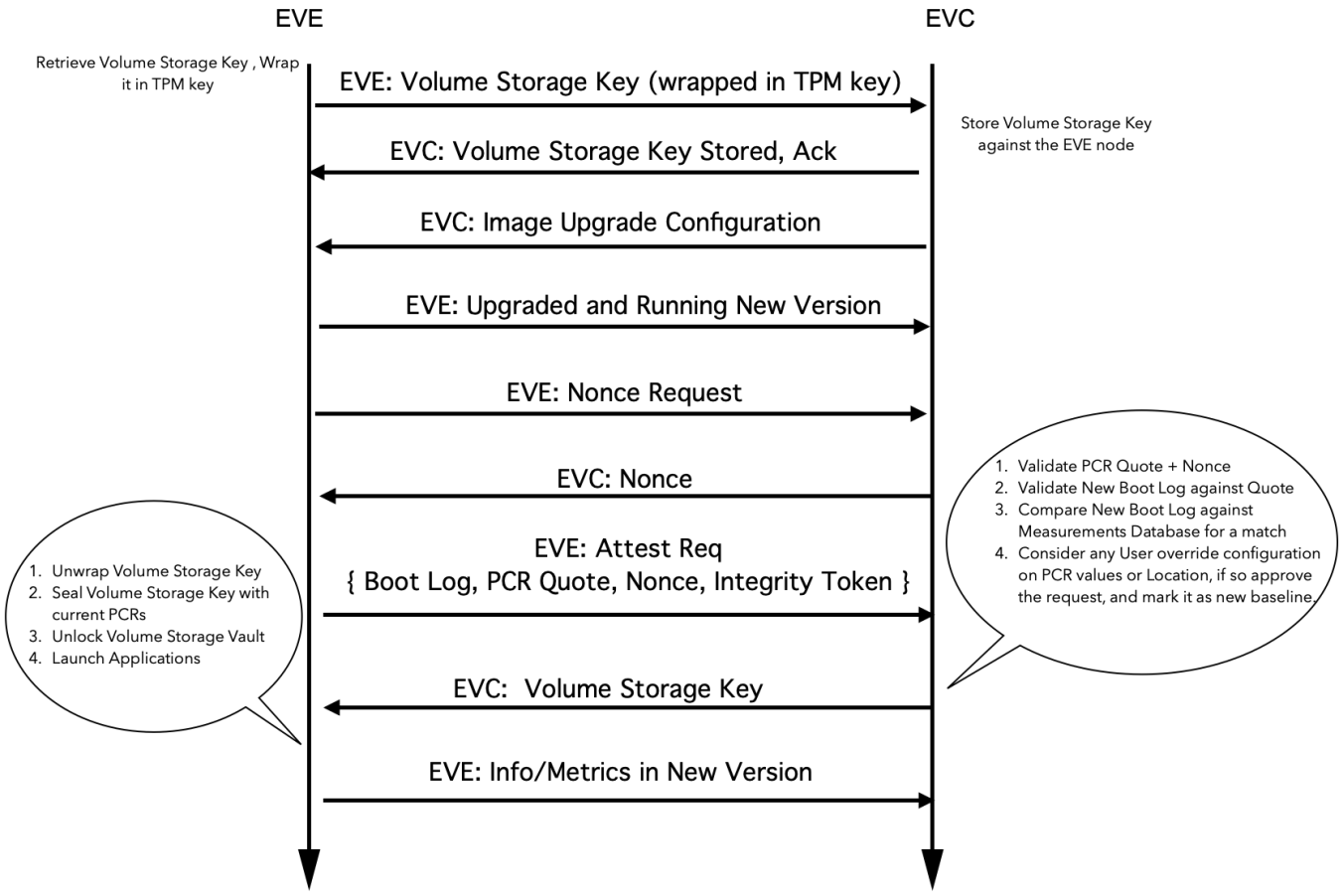

Fig 6. Overview of flow of events during Secure Software Update

Managing Firmware Upgrades

While the workflow is almost the same for securely upgrading firmware on EVE, the following additional steps are required:

1. BIOS/Firmware updates can be done either manually (say a person inserts USB drive with new firmware and updates it out-of-band, and reboots the device with new firmware) or can be done remotely (updates driven through remote management framework, without a physical access to the device). In both the cases, the new firmware version needs to be validated by EVC, before approving the new software state.
2. In case of remote updates, firmware Upgrades will be driven through EVC. This is to make sure EVC has the access to the firmware image, the measurement values to expect from the device for this firmware version etc.
3. To this effect, EVC would maintain a central database of all the BIOS firmware images, their signatures, and the certificate provided by the BIOS vendor for validating the signatures, indexed by a combined tag of BIOS version string + Manufacturer
4. EVC will communicate the upgrade to a dedicated agent for this purpose on EVE.
5. The Firmware upgrade agent can either be a EVE service or can be a service app deployed on EVE
6. The agent will have a platform abstraction layer and a platform dependent layer.
7. The platform abstraction layer drives the overall logic of downloading the firmware image, validating the image using manufacturer certificates, and reporting the status back to EVC
8. The platform dependent layer will be responsible for the actual firmware upgrade, by talking to any special component on the EVE hardware, like BMC
9. To address out-of-band update of firmware, there will be an option provided in the EVC portal, where the admin (after the upgrade is done), can configure EVC to accept the new firmware as trusted.
10. If this is a general update across many devices, admin can also configure EVC database to feed in the new hash values to expect from the new firmware version. Please note, this is almost the same as remote update workflow w.r.t. validating the new version, except that the image update is done out-of-band

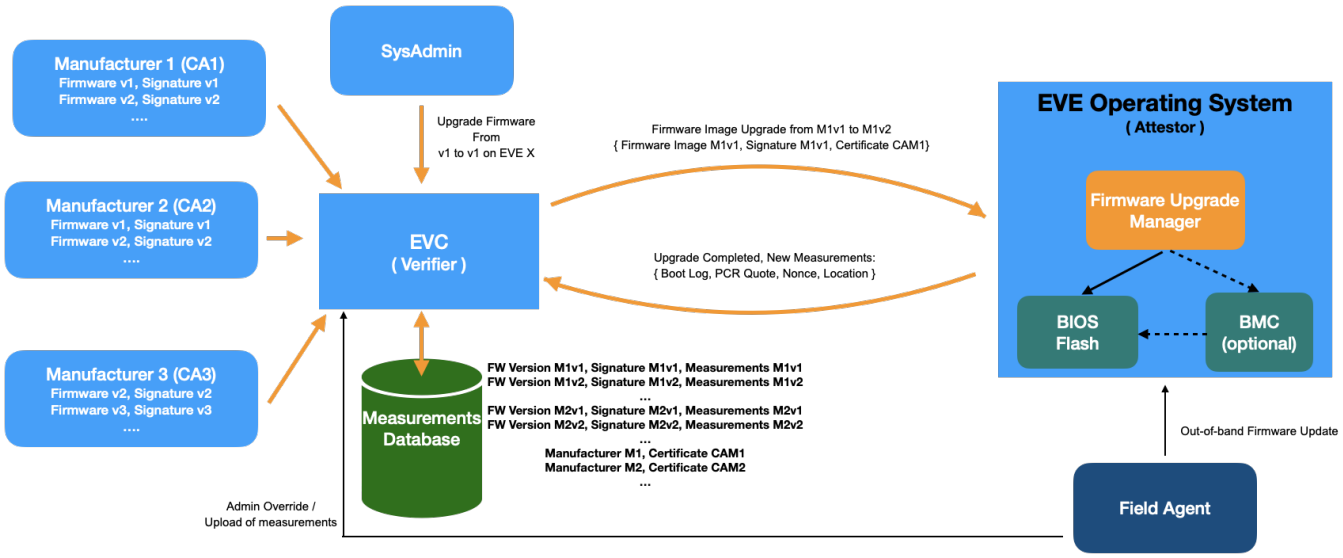

Fig 7. Firmware Upgrade Management

EVC Interface

Since there can be many implementations for EVC, to maintain compatibility, following API definitions are proposed for the purpose of implementing this feature:

URI	Type	Request Contains	Response Contains	Functionality
/api/v2/edgeDevice/uuid	POST	Empty payload, with just AuthContainer	UUID of the device or Error 404	Return UUID of the device, based on device certificate present in AuthContainer. So far, EVE has been re-using /api/v2/edgeDevice/config to get UUID of the device in the initial stages of starting EVE services. But with measured boot and remote attestation, configuration is sent to device only when device is attested for software integrity (which happens later as part zedAgent). Therefore, it may be observed that for getting UUID, fetching the whole configuration (which requires attestation now) would be an overkill, and hence the need for this new lightweight URI. The proposed /uuid is lightweight URI that responds with a single field that contains the UUID of the device. If the device is not known, Error 404 is sent.
/api/v2/edgeDevice/{uuid}/attestSubTypeATTEST_REQ_CERT	POST	X.509 Certificate signed by Device Certificate	No Response Payload, only the standard HTTP status codes	To send attestation certificate. This certificate is used to validate signature of the PCR Quote. This certificate is different from device certificate and ECDH certificate. The cert type for this will be set to CERT_TYPE_DEVICE_RESTRICTED_SIGNING. For more details, please refer this section of API documentation
/api/v2/edgeDevice/{uuid}/attestSubTypeATTEST_REQ_NONCE	POST	Empty payload, with just AuthContainer	Nonce Value	To request a nonce from controller. This nonce will be included while generating PCR Quote. This will trigger re-fetching of nonce from device. For more details, please refer this section of API documentation
/api/v2/edgeDevice/{uuid}/attestSubTypeATTEST_REQ_QUOTE	POST	PCR Quote, Event Log, Integrity Token, Image Version, Firmware Version, Nonce used	SUCCESS or FAILURE. If FAILURE Sub-code indicates whether there is a nonce mismatch. Also carries Integrity-Token Value, and the encrypted Volume Storage Key	To send { PCR Quote, Event Log, Integrity Token, and Image version } for attestation. If attestation result (PASS/FAIL) is sent back as response. If attestation is successful, EVC will cache the Integrity Token and the encrypted Volume Storage Key in the response. The next config request should contain the same Integrity-Token. Controller MUST generate NONCE_MISMATCH error if PCR quote is sent with a nonce that is not matching the nonce stored in EVC. For more details, please refer this section of API documentation

/api/v2/edgeDevice/{uuid}/config	POST	Hash of last configuration received and Integrity Token	Full configuration in case of proper Integrity Token, Error 403 if there is a Integrity-Token mismatch	Device will include Integrity-Token and hash of the last-received configuration in the request. If device presents Integrity-Token that matches copy in EVC, attestation is successful and full config is sent. If device is yet to attested successfully(indicated by an invalid or null Integrity Token), HTTP Error 403 is sent back, for device to trigger attestation first. Controller will run a periodic time check, and attestation has to be redone periodically at these intervals. EVC is free to choose a time value for this purpose. When attestation is due, EVC always indicates it to device via this 403 error code as response to config request.